UNIVERSITÀ PARTHENOPE
DiST
Dipartimento di Scienze e Tecnologie

Intelligent Signal Processing

Brief Introduction to Python

Angelo Ciaramella

Eccezioni

- ArithmeticError
  - Exceptc catturerà ArithmeticError
    - ma anche i suoi tre sotto-tipi ZeroDivisionError, OverflowError e FloatingPointError

```
-------------------------------------------------------
# script
try:
    n = 5/0
except ArithmeticError:
        print('Invalid operation!')
-------------------------------------------------------
Console
-------
>>> Invalid operation!
```

Esempiod di eccezione ArithmeticError

Eccezioni

```
-----------------------------------------------------
# script
try:
    n = int(x) # prova a convertire x in intero ...
except ValueError:
    # eseguito in caso di ValueError
    print('Invalid number!')
except TypeError:
    # eseguito in caso di TypeError
    print('Invalid type!')
else:
    # eseguito se non ci sono errori
    print('Valid number!')
```

Uso di else nelle eccezioni

Funzioni

```
-------------------------------------------------------------
# script
f = open('test.txt', 'w') # apre un file in scrittura
try:
    f.read() # prova a leggere e fallisce
finally:
    f.close() # il file viene chiuso

-------------------------------------------------------------
Console
-------
>>> f.closed
```

Clausola finally

Lanciare un'eccezione

```
-------------------------------------------------------
# script
def div(num, den):
    if den == 0:
        # se il denominatore è 0 riporta un'eccezione
        raise ZeroDivisionError('Impossibile dividere per
0')
    return num / den

-------------------------------------------------------
Console
-------
>>> div(8,0)
```

Lanciare un'eccezione con raise

N.B.: E' possibile avere sottoclassi di eccezioni

Gestione dei File

Modalità	Descrizione
'r'	Apre un file di testo in lettura. Modo di apertura di default dei file.
'w'	Apre un file di testo in scrittura. Se il file non esiste lo crea, altrimenti cancella il contenuto del file.
'a'	Apre un file di testo in *append*. Il contenuto viene scritto alla fine del file, senza modificare il contenuto esistente.
'x'	Apre un file di testo in creazione esclusiva. Se il file non esiste, restituisce un errore, altrimenti apre in scrittura cancellando il contenuto del file.
'r+'	Apre un file di testo in modifica. Permette di leggere e scrivere contemporaneamente.
'w+'	Apre un file di testo in modifica. Permette di leggere e scrivere contemporaneamente. Cancella il contenuto del file.

Gestione file

```
-------------------------------------------------------------
# script
f = open(`./test.txt', 'w')

-------------------------------------------------------------
Console
-------
>>> div(8,0)
```

Lanciare un'eccezione con raise

N.B.: E' possibile avere sottoclassi di eccezioni

Gestione dei File

metodo	Descrizione
file.read()	Legge e restituisce l'intero contenuto del file come una singola stringa.
file.read(n)	Legge e restituisce n caratteri (o byte).
file.readline()	Legge e restituisce una riga del file.
file.readlines()	Legge e restuisce l'intero contenuto del file come lista di righe (stringhe).
file.write(s)	Scrive nel file la stringa s e ritorna il numero di caratteri (o byte) scritti.
file.writelines(lines)	Scrive nel file la lista in righe lines.
file.tell()	Restituisce la posizione corrente memorizzata dal file object.
file.seek(offset, pos)	Modifica la posizione corrente memorizzata dal file object.
file.close()	Chiude il file.

Gestione file (with)

```
-------------------------------------------------------
# script
with open('./test.txt', 'w') as f:
     f.write('contenuto del file')

# chiude automaticamente il file

-------------------------------------------------------
Console
-------
>>> f.closed
```

Utilizzo di with

Moduli

```
-------------------------------------------------------------
# script
import math           # import della libreria matematica

-------------------------------------------------------------
Console
-------
>>> help(math)
```

```
# script
from math import pi, sqrt # import di contenuti specifici
```

```
# script
import math as matematica # assegna un nuovo nome
from math import sqrt as radice_quadrata
```

Caricamento dei moduli

Alcuni moduli standard

Modulo	Descrizione
re	Espressioni regolari
datetime	Date e ore
collections	Oggetti contenitori
enum	Enumerazioni
decimal	Aritmetica in virgola mobile
random	Numeri pseudo-casuali
statistics	Funzioni statistiche
itertools	iteratori
csv	Supporto file csv

Alcuni moduli standard

Modulo	Descrizione
json	Supporto per file JSON
email	Supporto per email
html, xml	Supporto per HTML e XML
os	System call
io	File e stream
socket	Interfaccia di rete
tkinter	Creazione di GUI
sys	Parametri di sistema
zipfile	Archiviazione zip

Moduli

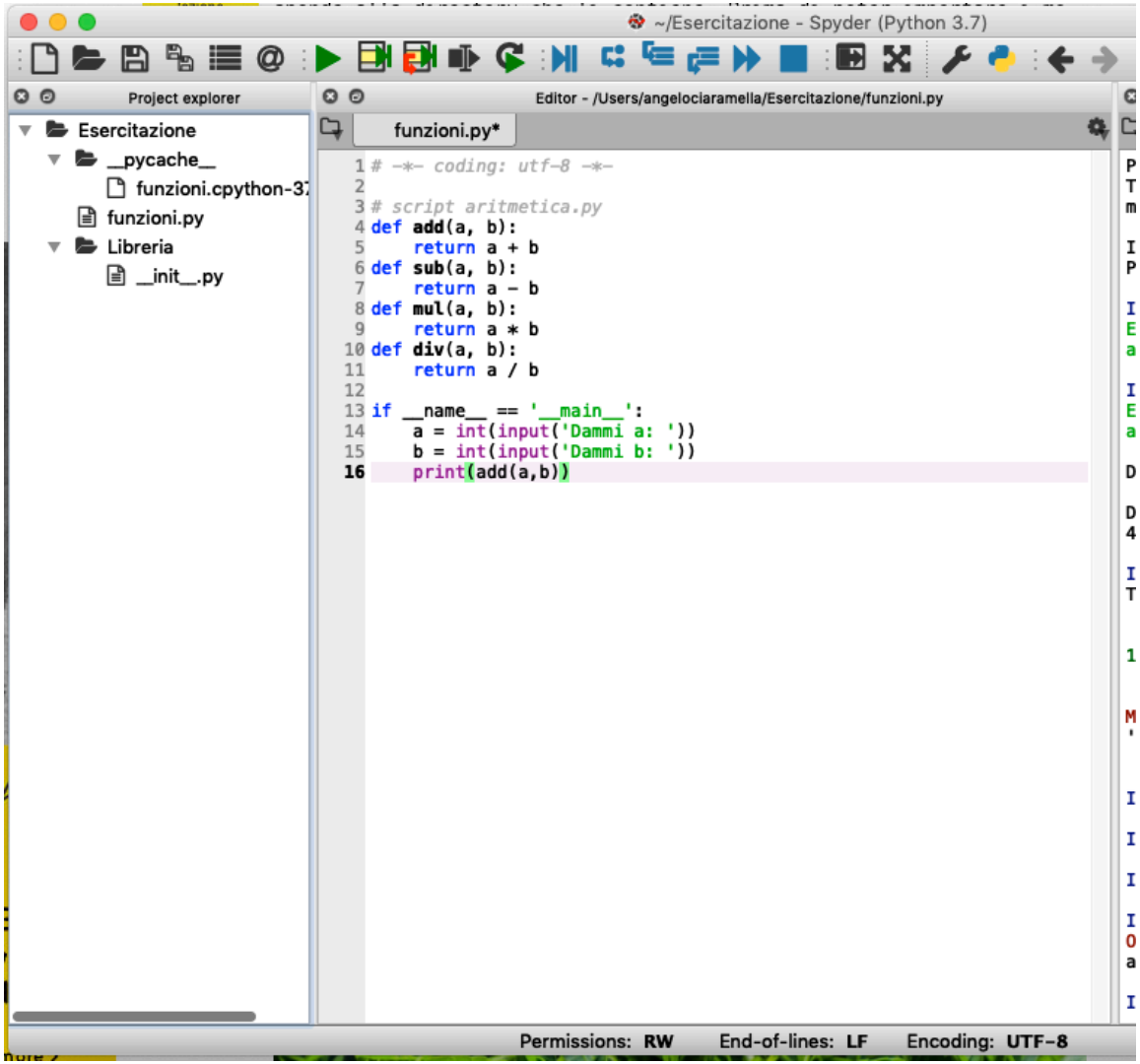
```
# script aritmetica.py
def add(a, b):
    return a + b
def sub(a, b):
    return a - b
def mul(a, b):
    return a * b
def div(a, b):
    return a / b

if __name__ == '__main__':
    a = int(input('Dammi a: '))
    b = int(input('Dammi b: '))
    print(add(a,b))
```

Esempio di creazione di modulo

Provare ad eseguirlo e a fare l'import

Package



Nel Package per l'import inserire in ogni sottocartella il file vuoto __init__py

Tono puro

```
# script aritmetica.py
import numpy as np
import sounddevice as sd
import matplotlib.pyplot as plt

# Signal frequency
frequency = 440

# Amplitude
amplitude = 100

# Sampling frequency
sampling_rate = 44100

# Seconds of the sequence
sec = 1

# Number of samples
num_samples = sec * sampling_rate
```


Tono puro

```
# time
t = np.arange(0,num_samples-1)

# sinusoidal wave
sine_wave = amplitude*np.sin(2 * np.pi * frequency * t /
sampling_rate)
# sine_wave = [amplitude* np.sin(2 * np.pi * frequency * t /
sampling_rate) for t in range(num_samples)]

# Function plot
plt.plot(t/sampling_rate, sine_wave, color='blue',
label="tono puro")
plt.title('Tono Puro')
plt.xlabel('t')
plt.ylabel('y(t)')
plt.legend()
plt.show()


# Sound of the signal
sd.play(sine_wave,sampling_rate)
sd.stop()
```


Programmazione OO

```
-------------------------------------------------------
# definizione di classe
class Rectangle:
    def __init__(self, base, height): # costruttore
        self.base = base          # attributi
        self.height = height
    def calc_area(self):
        return self.base * self.height
    def calc_perimeter(self):
        return (self.base + self.height) * 2

-------------------------------------------------------
Console
-------
>>> myrect = Rectangle(3, 5) #istanza della classe
>>> myrect.base              # chiamata di un attributo
>>> myrect.calc_area()       # chiamata di un metodo
```

Esempio di costruzione e utilizzo di classe

Le classi

```
-------------------------------------------------------
# definizione di classe
class Dog:
    scientific_name = 'Canis lupus familiaris'
    def __init__(self, name):
        self.name = name
-------------------------------------------------------
Console
-------
>>> rex = Dog('Rex')
>>> fido = Dog('Fido')
>>> Dog.scientific_name
>>> rex.scientific_name
>>> fido.scientific_name
```

Attributi di classe

Ereditarietà

```
-------------------------------------------------------
# definizione di classe
class Person:
    def __init__(self, name, surname):
        self.name = name
        self.surname = surname
    def eat(self, food):
        print(self.name, 'is eating', food)
    def sleep(self):
        print(self.name, 'is sleeping')
-------------------------------------------------------
Console
-------
>>> p = Person('Nome', 'Cognome')
```

Definizione di classe per l'ereditarietà

Ereditarietà

```
-------------------------------------------------------
# definizione di classe
class Employee(Person):
    def __init__(self, name, surname, job):
        super().__init__(name, surname)
        self.job = job
    def work(self):
        print(self.name, 'is working as a', self.job)
-------------------------------------------------------
Console
-------
>>> e = Employee('Nome', 'Cognome', 'developer')
```

Esempio di ereditarietà singola

N.B.: in Python è possibile l'ereditarietà multipla – class Employee(Person, Worker, …)

Metodi speciali

```
-------------------------------------------------------
# definizione di classe
class Person:
    . . .

    def __str__(self):
        return '{} {}'.format(self.name, self.surname)
    def __repr__(self):
        return '<Person object ({} {})>'.format(self.name,
                  self.surname)

-------------------------------------------------------
Console
-------
>>> p = Person('Nome', 'Cognome')
>>> str(p)
>>> repr(p)
>>> print(p)
```

N.B.: in Python è possibile fare l'oerload degli operatori

Attributi e metodi privati

```
# definizione di classe
class Priv(object):
    def __init__(self):
        self.__var = 42
    def get(self):
        return self.__var
    def __get_private(self):
        print("Metodo privato")

-------------------------------------------------------
Console
-------
>>> p = Priv()
>>> p.__var
>>> p.__get_provate()
```

Esempio di variabili e metodi privati

Iteratori

```
# definizione di classe
class Reverse:
     def __init__(self, data):
          self.data = data
          self.index = len(data)
     def __iter__(self):
          return self
     def __next__(self):
          if self.index == 0:
             raise StopIteration
          self.index = self.index - 1
          return self.data[self.index]

-----------------------------------------------------
Console
-------
>>> rev = Reverse('spam')
>>> iter(rev)
>>> for char in rev:
...      print(char)
```


Classe Astratta

```
class Animal:
     def move(self):
         pass

class Human(Animal):
    def move(self):
        print("Mi posso muovere e correre ...")

class Snake(Animal):
    def move(self):
        print("Posso strisciare ...")

def main():
    a1 = Animal()
    h1 = Human()
    h1.move()
    s1 = Snake()
    s1.move()

if __name__ == '__main__':
    main()
```


Classe Astratta

```
from abc import ABC, abstractmethod

class Animal(ABC):

    @abstractmethod
    def move(self):
        pass

. . .
```

Esempio di annotazione